Perform multi-site water balance simulations using LWFBrook90R

Paul Schmidt-Walter

2021-01-28

Contents

Introduction

With the new LWFBrook90R version 0.4.0, multisite-simulations are becoming more flexible, through new possibilities for retrieving climatic data input from a function, and to use the `output_fun` argument of `run_LWFB90()` with reference to identifiers for soil, climate and parameters, when called from `run_multisite_LWFB90()`. To demonstrate the potential of the new functionality, multi run simulations will be performed that reproduce the data of the NFIWADS database, containing monthly and yearly aggregates of water fluxes and soil moisture state variables for the National Forest Inentory of Germany (NFI).

Prerequisites and data

To reproduce the shown examples, please install LWFBrook90R version 0.4.0 or higher. You can install the package directly from github.com using the package 'remotes':

```
remotes::install_github(repo="pschmidtwalter/LWFBrook90R",
                        ref = "v0.4.0",
                        build_vignettes=TRUE)
```

Now load the required packages:

```
library(LWFBrook90R)
library(data.table)
library(RSQLite)
```

The NFIWADS sample data is available as supplementary materials of NFIWADS-publication at ZENODO. We download the .zip file to a temporary location and extract the sqlite database file 'NFIWADS_input.sqlite', that contains the input along with a sample of climatic data, to save it in the working directory:

```
tdir <- tempdir()
tfile <- tempfile(tmpdir = tdir, fileext = ".zip")
download.file("https://zenodo.org/record/1491520/files/NFIWADS_supplementary_material.zip", tfile)
unzip(tfile,  exdir= tdir, overwrite=TRUE)
```

```
file.copy(file.path(tdir, "NFIWADS_input.sqlite"), getwd())
#> [1] FALSE
```

Before running the NFIWADS example, we will also run some small examples using the sample data that comes with LWFBrook90R. Therefore, we load the climatic and soil physical data, create standard `param_b90` and `options_b90` objects and set up a `soil` data.frame, using a pedotransfer function:

```
data("slb1_soil", "slb1_meteo")
opts <- set_optionsLWFB90(startdate = as.Date("2002-06-01"), enddate = as.Date("2002-06-02"))
parms <- set_paramLWFB90()
soil <- cbind(slb1_soil, hydpar_wessolek_tab(texture = slb1_soil$texture))
```

Multi-site simulations: climatic data input from a function

A basic multi-site simulation using the function `run_multisite_LWFB90()` runs through lists of `param_b90`, `climate`, and `soil`-objects, and evaluates the specified parameter sets for each of the soil/climate combinations:

```
test <- run_multisite_LWFB90(options_b90 = opts,
               param_b90 = list(beech = parms,spruce = parms),
               climate = slb1_meteo,
               soil = list(soil1 = soil, soil2 = soil, soil3 = soil))
str(test, max.level = 1)
#> List of 6
#>  $ soil1 beech :List of 5
#>  $ soil1 spruce:List of 5
#>  $ soil2 beech :List of 5
#>  $ soil2 spruce:List of 5
#>  $ soil3 beech :List of 5
#>  $ soil3 spruce:List of 5
```

The results are returned as a list of the single run objects, as returned by `run_LWFB90()`. The list entries are named according to the names of the list entries holding the individual `param_b90`, `climate`, and `soil` input objects. The function can easily be set up to run the 10 samples from the NFIWADS input database for which climatic data is available. However, simulating all sites from NFIWADS database in this way is not possible, because such a large list of `climate` data.frames does not fit in the memory. Fortunately, it is possible to pass a function instead of a `data.frame` as `climate`-argument to `run_LWFB90()`. Such a function can be used to create the `climate`-data.frame from a file or database-connection within `run_LWFB90` or `run_multisite_LWFB90()`. For `run_LWFB90`, we can provide arguments to the function simply via the `...`-placeholder. For `run_multisite_LWFB90()`, however, we need to pass arguments to the function (possibly with individual values for individual site, e.g. a file name) via the `climate_args`-argument.

To demonstrate this mechanism, we set up the database connection to 'NFIWADS_input.sqlite', extract the site ids for which climatic data is available and display the tables in the database:

```
#set up connection
dbinput <- dbConnect(SQLite(), "NFIWADS_input.sqlite")

# list tables
dbListTables(dbinput) #list tables
#> [1] "climate_daily_int100_sample"
#> [2] "echam6_tmean_stars24_stnlauf_rcp26_lauf79"
#> [3] "echam6_tmean_stars24_stnlauf_rcp45_lauf98"
#> [4] "echam6_tmean_stars24_stnlauf_rcp85_lauf89"
#> [5] "location_parameters"
#> [6] "soils"
```

```
#> [7] "z_col"
#> [8] "z_tab"

#retrieve site_ids for which climatic data is available
ids_clim <- dbGetQuery(dbinput, "select distinct id from climate_daily_int100_sample")$id
```

The climatic data resides in the table 'climate_daily_int100_sample'. All variables are stored as integers and need to be divided by 100 for converting them to the units required by `run_LWFB90()`. Accordingly, we define a function, that retrieves the climatic data for a specific NFI site from a table in the database, does the data processing and returns a suitable `climate` data.frame:

```
climfun <- function(con, tbl, site_id){
  #get data from connection con
  qry <- paste0("select * from ", tbl, " where id = '", site_id, "';")
  clim <- data.table(dbGetQuery(con, qry))

  # process data to be usable as 'climate' in LWFBrook90R::run_LWFB90()
  # use suitable names
  setnames(clim, c("grhds", "rrds","sddm","tadm", "tadn", "tadx","wsdm"),
           c("globrad", "prec","vpd","tmean", "tmin", "tmax","windspeed"))
  # convert units
  clim[, c("globrad", "prec","vpd","tmean", "tmin", "tmax","windspeed") :=
         list(globrad/100, prec/100,vpd/100,tmean/100, tmin/100, tmax/100,windspeed/100)]

  # create date variable
  clim[,dates := as.Date(paste(year, month, day, sep="-"))]

  # calculate vappres from vpd
  clim[, es :=  ifelse(tmean >= 0,
                       6.1 * 10^( (7.5 * tmean) / (tmean + 237.2) ) ,
                       6.1 * 10^( (9.5*tmean) / (tmean + 265.5) ) )]
  clim[, vappres := ifelse( (es - vpd) <0, 0, (es - vpd)*0.1)] #vpd und es in hPa

  # return sorted
  clim[order(dates),list(dates, tmin, tmax, tmean, globrad, vappres, prec, windspeed)]
}
```

Lets test our function with the database connection set up earlier to see that it works:

```
head(climfun(con = dbinput, tbl = "climate_daily_int100_sample", site_id = ids_clim[1]))
#>         dates  tmin tmax tmean globrad   vappres prec windspeed
#> 1: 1961-01-01 -1.20 1.30  0.23    1.97 0.5772905 2.81      2.04
#> 2: 1961-01-02 -1.32 1.08  0.45    2.00 0.5892769 3.83      3.33
#> 3: 1961-01-03  1.32 5.01  2.40    2.04 0.6511954 5.77      4.55
#> 4: 1961-01-04  0.08 4.42  1.80    3.22 0.5877286 0.62      3.58
#> 5: 1961-01-05  0.62 2.72  1.08    2.88 0.5806650 0.54      4.94
#> 6: 1961-01-06 -2.23 1.60  0.38    2.27 0.5520841 1.42      4.78
```

Before we run our multi-site simulation, we need to set up lists of arguments for our above defined function, one for each site, and store them together in a list. We define a list for each site, holding the required arguments. Although `site_id` is the only site-dependend parameter, we need to specify all non-default arguments for each site here. We loop across our earlier retrieved vector of ids and name the entries according to the site ids, so they will be propagated to the output of the simulations.

```
clim_args <- lapply(ids_clim, function(x) {
  list(con = dbinput,
```

```
      tbl = "climate_daily_int100_sample",
      site_id = x)
  })
names(clim_args) <- ids_clim
```

For the simulation, we need to define the model control options, have to set up the list of soil `data.frames`, and create the two parameter sets that we want to run:

```
# create LWFB90 options
opts <- set_optionsLWFB90(startdate = as.Date("2000-01-01"),
                          enddate = as.Date("2010-12-31"),
                          root_method = "soilvar",
                          budburst_method = "Menzel",
                          leaffall_method = "vonWilpert")

# retrieve soil data as data.table for ids_clim from database
soils <- data.table(
  dbGetQuery(dbinput, paste0("select * from soils where id in (",
                             paste(paste0("'",ids_clim,"'"), collapse = ", "),
                             "); "))
)
# convert to data.table and split to list
soils_list <- split(soils, by = "id")

# create NFIWADS parameter sets
param_beech <- set_paramLWFB90(winlaifrac = 0.1,maxlai = 6,sai = 1,height = 30,
                               frintlai = 0.1,frintsai = 0.25,cintrl = 0.06,cintrs = 0.4,
                               fsintlai = 0.1,fsintsai = 0.6,cintss = 0.6,cintsl = 0.6,
                               alb = 0.18,albsn = 0.23,lwidth = 0.05,glmax = 0.0042,
                               budburst_species = "Fagus sylvatica",infexp = 0.66)

param_spruce <- set_paramLWFB90(winlaifrac = 0.8,maxlai = 5.5,sai = 1,height = 30,
                                frintlai = 0.12,frintsai = 0.14,cintrl = 0.2,cintrs = 0.4,
                                fsintlai = 0.12,fsintsai = 0.14,cintss = 0.6,cintsl = 0.6,
                                alb = 0.14,albsn = 0.14,lwidth = 0.004,glmax = 0.0035,
                                budburst_species = "Picea abies (frueh)",infexp = 0.66)
```

Now we can run the multi-site simulation across the ten sites, using the two above defined parameter sets. The list of `soil` data.frames is created by splitting the `soils` according to their site id:

```
test2 <- run_multisite_LWFB90(options_b90 = opts,
               param_b90 = list(beech = param_beech,spruce = param_spruce),
               climate = climfun,
               climate_args = clim_args,
               output = -1,
               soil = soils_list,
               cores = 5)
str(test2, max.level = 1)
#> List of 20
#>  $ 12126_2 12126_2 beech :List of 5
#>  $ 12126_2 12126_2 spruce:List of 5
#>  $ 13465_4 13465_4 beech :List of 5
#>  $ 13465_4 13465_4 spruce:List of 5
#>  $ 13803_1 13803_1 beech :List of 5
#>  $ 13803_1 13803_1 spruce:List of 5
```

```
#>  $ 14665_2 14665_2 beech :List of 5
#>  $ 14665_2 14665_2 spruce:List of 5
#>  $ 1532_1 1532_1 beech   :List of 5
#>  $ 1532_1 1532_1 spruce  :List of 5
#>  $ 15883_3 15883_3 beech :List of 5
#>  $ 15883_3 15883_3 spruce:List of 5
#>  $ 16541_4 16541_4 beech :List of 5
#>  $ 16541_4 16541_4 spruce:List of 5
#>  $ 17346_2 17346_2 beech :List of 5
#>  $ 17346_2 17346_2 spruce:List of 5
#>  $ 17565_1 17565_1 beech :List of 5
#>  $ 17565_1 17565_1 spruce:List of 5
#>  $ 20075_4 20075_4 beech :List of 5
#>  $ 20075_4 20075_4 spruce:List of 5
```

Multi-Site simulations: redirect output to file or database

The example above showed how to avoid loading climatic data into the workspace all at once, by evaluating a function on-the-fly to deliver and process the data. However, the output of `run_multisite_LWFB90()` can also become very large if many sites are simulated, and the returned data is selected to contain daily values. To reduce the output of the single run simulations, it is possible to provide functions for `run_LWFB90()` via its `output_fun`-argument, that perform directly on the output of simulation. These can be used to do some custom aggregations or write the results to a file, while suppressing the return of the regular selected output (as specified by `output`-argument) using `rtrn_output = F`. In `run_multisite_LWFB90()`, such output-functions and further arguments can be easily passed via the `...` placeholder to `run_LWFB90()`. However, `run_LWFB90()` until recently did not know anything about the calling `run_multisite_LWFB90()` and the names of the parameter/soil/climate data sets (as specified by the names of the input list entries) it was currently processing. This was a problem, if the output function was defined to write its results to a file, because no identifier for the climate/soil/parameters sets was available inside `run_LWFB90()`. Since version 0.4.0, the names of the current soil, climate, parameter objects are passed automatically from `run_multisite_LWFB90()` to `run_LWFB90()` as objects `soil_nm`, `clim_nm`, and `param_nm`. In this way, they are accessible to `output_fun`-functions within `run_LWFB90()`.

To demonstrate this mechanism, create a folder 'output', and define a simple function, that will writes the results of the `daily_output` objects of the individual runs to files in that folder. The file names are concatenated from the names of the climate, soil, and parameter-sets.

```
dir.create("output")
#> Warning in dir.create("output"): 'output' existiert bereits
outfun <- function(x, clim_nm, soil_nm, param_nm) {
  fname = paste0("output/output_", clim_nm,"_",soil_nm, "_", param_nm, ".csv")
  write.csv(x$daily_output, file = fname)
}
```

Now we can run the multi-site simulation as before, but suppress the regular output of `run_LWFB90()` using `rtrn_output=FALSE`, and pass above defined function:

```
test3 <- run_multisite_LWFB90(options_b90 = opts,
               param_b90 = list(beech = param_beech,spruce = param_spruce),
               climate = climfun,
               climate_args = clim_args,
               soil = soils_list,
               rtrn_input = FALSE,
               rtrn_output = FALSE,
```

```
                output_fun = outfun,
                cores = 5)
```

Let's take a look to one of the single run results in the returned list:

```
test3[1]
#> $`12126_2 12126_2 beech`
#> $`12126_2 12126_2 beech`$simulation_duration
#> Time difference of 2.540137 secs
#>
#> $`12126_2 12126_2 beech`$finishing_time
#> [1] "2021-01-28 14:35:07 CET"
#>
#> $`12126_2 12126_2 beech`$output_fun
#> $`12126_2 12126_2 beech`$output_fun[[1]]
#> NULL
```

We see that the entries do not contain any simulation results, because we passed `rtrn_output = FALSE` to `run_LWFB90()`. As well, the entry `output_fun` is `NULL`, because `write.csv` has no return value when a file is specified. We can see however that out attempt was successful, the simulation results were written to our 'output'-directory:

```
list.files("output")
#>  [1] "nfiwads_results.sqlite"            "output_12126_2_12126_2_beech.csv"
#>  [3] "output_12126_2_12126_2_spruce.csv" "output_13465_4_13465_4_beech.csv"
#>  [5] "output_13465_4_13465_4_spruce.csv" "output_13803_1_13803_1_beech.csv"
#>  [7] "output_13803_1_13803_1_spruce.csv" "output_14665_2_14665_2_beech.csv"
#>  [9] "output_14665_2_14665_2_spruce.csv" "output_1532_1_1532_1_beech.csv"
#> [11] "output_1532_1_1532_1_spruce.csv"   "output_15883_3_15883_3_beech.csv"
#> [13] "output_15883_3_15883_3_spruce.csv" "output_16541_4_16541_4_beech.csv"
#> [15] "output_16541_4_16541_4_spruce.csv" "output_17346_2_17346_2_beech.csv"
#> [17] "output_17346_2_17346_2_spruce.csv" "output_17565_1_17565_1_beech.csv"
#> [19] "output_17565_1_17565_1_spruce.csv" "output_20075_4_20075_4_beech.csv"
#> [21] "output_20075_4_20075_4_spruce.csv"
```

Our output function did not use any external arguments, all information (data and names of the current input objects to create a file name) was internally generated within `run_multisite_LWFB90()`. Similarily, we could have written the simulation results to a database. To do so, we just needed to set up a database connection inside the function, and close it afterwards. Arguments for setting up the connection can simply be passed to `output_fun` via the `...` place-holder in the call of `run_multisite_LWFB90()`. We will learn how to do this in the next section.

Reproduction of NFIWADS database

Now that we showed that our simple `output_fun` function worked, we can create a more complex function that calculates the aggregates of soil moisture status variables and water fluxes for the NFIWADS database. The output created consists of two tables containing monthly aggregates and aggregates for the vegetation period. To reduce code of the function, we outsourced some of the code in functions that can be found in the directory 'R-Functions'.

We source the functions inside our function call (see below). The first function (`aggr_swati`) aggregates the daily layer-wise soil moisture and fluxes variables (`layer_output`-data.frame) to daily values, while the other functions aggregate daily values to monthly or vegetation period representations, for which the beginning and end day-of-year of the vegetation period have to be specified as vectors covering the years of the simulation. The aggregation of the water fluxes simply take the output object `daily_output`, while soil

moisture state aggregation functions are designed to use the returned daily values from the first function `aggr_swati`. Referring to these functions, we can define our `output_fun` to combine the results in two tables, and write them to two different tables (>70 columns!) in a database:

```
outfun_nfiwads <- function(x, clim_nm, soil_nm, param_nm, db_path) {

  # source helper functions
  source("R-Functions/aggr_swati.R", local = T)
  source("R-Functions/aggr_fluxes_daymon.R", local = T)
  source("R-Functions/aggr_states_soil_daymon.R", local = T)
  source("R-Functions/aggr_fluxes_vegper.R", local = T)
  source("R-Functions/aggr_states_soil_vegper.R", local = T)

  # gather variables from model-input for later use
  vpstart <- x$model_input$param_b90$budburstdoy
  vpend <- x$model_input$param_b90$leaffalldoy

  soil <- with(x$model_input$param, merge(soil_nodes, soil_materials, by = "mat"))

  swat.profile <- aggr_swati(x$layer_output, soil) # aggregate layer output

  out_monthly <- data.table::data.table(site_id = clim_nm, parset = param_nm,
                           fluxes_dailytomonthly(x$daily_output),
                           states_soil_daymon(swat.profile)[,-c(1,2), with = F])
  data.table::setnames(out_monthly, names(out_monthly), tolower(names(out_monthly)))

  out_vegper <- data.table::data.table(
    site_id = clim_nm, parset = param_nm,
    fluxes_dailytovegper(x$daily_output,vp.start = vpstart, vp.end = vpend),
    states_soil_dayvp(swat.profile, vp.start = vpstart, vp.end = vpend)[,-c(1,2), with = F])
  data.table::setnames(out_vegper, names(out_vegper), tolower(names(out_vegper)))

  # write to db
  # open connection
  db_out <- DBI::dbConnect(RSQLite::SQLite(), db_path)
  if (DBI::dbExistsTable(db_out, "nfiwads_monthly")) {
    chk_m <-DBI::dbAppendTable(db_out, "nfiwads_monthly", out_monthly)
  } else {
    DBI::dbCreateTable(db_out, "nfiwads_monthly", out_monthly)
    chk_m <-DBI::dbAppendTable(db_out, "nfiwads_monthly", out_monthly)
  }

  if (DBI::dbExistsTable(db_out, "nfiwads_vegper")) {
    chk_vp <- DBI::dbAppendTable(db_out, "nfiwads_vegper", out_vegper)
  } else {
    DBI::dbCreateTable(db_out, "nfiwads_vegper", out_vegper)
    chk_vp <- DBI::dbAppendTable(db_out, "nfiwads_vegper", out_vegper)
  }
  #close
  DBI::dbDisconnect(db_out)

  if (chk_vp & chk_m) {
    return("Success!1!!")
  }
```

```
}
```

The database should be created first. Like the input database, we use an SQLite file, which is created when setting up the connection:

```
db_out <- dbConnect(SQLite(), "output/nfiwads_results.sqlite")
```

We can test our function with one of the single runs from our `test2` multirun simulation to see that it works:

```
outfun_nfiwads(test2[[1]],
           clim_nm = "clim", soil_nm = "soil", param_nm = "test",
           db_path = "output/nfiwads_results.sqlite")
#> [1] "Success!1!!"
```

The database now contains two tables, populated with data:

```
dbListTables(db_out)
#> [1] "nfiwads_monthly" "nfiwads_vegper"
#remove test-tables
dbRemoveTable(db_out, "nfiwads_vegper")
dbRemoveTable(db_out, "nfiwads_monthly")
```

Now it is time to run the NFIWADS sample, after having cleared the previously created test-tables. Again, we set `rtrn_output = F`, and also `rtrn_input = F`, so that only some small piece of information about the runs are returned. Note that even though the model input is not appended to the simulation results, it is available to our `output_fun`-function (`x`-argument), where we extract information on the beginning and end of the vegetation period and information on soil data.

```
test4 <- run_multisite_LWFB90(
  options_b90 = opts,
  param_b90 = list(beech = param_beech,spruce = param_spruce),
  climate = climfun,
  climate_args = clim_args,
  soil = soils_list,
  rtrn_input = F,
  rtrn_output = F,
  output_fun = outfun_nfiwads,
  db_path = "output/nfiwads_results.sqlite",
  cores = 5)
```

Die Simulation läuft sauber durch, jedoch gibt es auf meinem REchner ein Warning, dass parallel erzeugte Zufallszahlen nicht unabhängig von einandern sind. Wahrscheinlich hat das etwas mit den Datenbankverbindungen zu tun, Zufalleszahlen werden in LWFBrook90R jedenfalls nicht erzeugt. Mit folgendem Befehl lässt sich die Warnung abschalten:

```
options(future.rng.onMisuse = 'ignore')
```